

Etape 12 : Partage entre Thread

Multithreading en JAVA

Partage entre Thread

- ▶ Le thread s'exécute dans un même espace d'adressage alloué au processus les supportant :
 - Il est donc possible de partager des variables entre ces threads
 - plusieurs façons de faire :
 - *des variables static* => toutes les instances accèdent à la même adresse
 - *des références communes* => créées par un thread et données aux autres
- ▶ Lorsque plusieurs Thread véhiculent la même instance de Runnable c'est aussi une forme de partage

Partage entre Thread

► cas de la variable *static* :

- les thread Dalton partagent une donnée commune : le Mago

```
class Dalton extends Thread {  
  
    static Tresor LeMagot;  
    private String prenom;  
  
    public void run() {  
        Random rand = new Random();  
        int nbBilletsDansPoche = rand.nextInt(500) + 500;  
  
        // je vide mes poches et j'ajoute au mago !  
        for (int i = 1; i <= nbBilletsDansPoche ; i++) {  
            LeMagot.ajouterAuMago(1);  
            System.out.println "[" + prenom + " ]:" + LeMagot.getMontant();  
        }  
    }  
}
```

Partage entre Thread

► cas de la variable *static* : on lance le décompte du dernier casse des thread Dalton :

```
public static void main(String[] args) {  
  
    Dalton.LeMagot=new Tresor(0);  
  
    Thread jo = new Dalton("Jo");  
    Thread averell = new Dalton("Averell");  
  
    jo.start();  
    averell.start();  
}
```

Partage entre Thread

- cas de la variable *static* : la donnée partagée

```
class Tresor {  
    private Integer montant;  
  
    public Tresor(Integer montant)  
    { this.montant=montant; }  
  
    public void ajouterAuMago(Integer somme)  
    { this.montant+=somme; }  
  
    public Integer getMontant()  
    { return this.montant; }  
  
}
```

Partage entre Thread

- on va pouvoir constater que Jo et Averell coopèrent et incrémentent l'objet partagé :

```
[Averell] je vide mes poches et j'ajoute au mago ! 763  
[Jo] je vide mes poches et j'ajoute au mago ! 804  
[Averell ]:0  
[Jo ]:1  
[Jo ]:2  
[Averell ]:3  
[Jo ]:4  
[Averell ]:5  
[Jo ]:6  
...  
...  
[Jo ]:1564  
[Jo ]:1565  
[Jo ]:1566  
[Jo ]: J'ai fini !!
```

Partage entre Thread : partage d'une référence

Sans utiliser un attribut *static*, on peut aussi partager une référence vers une variable d'instance :

```
public static void main(String[] args) {  
    Butin leMago=new Butin(0);  
    Thread jo = new Dalton("Jo",leMago);  
    Thread averell = new Dalton("Averell",leMago);  
    jo.start();  
    averell.start();  
}
```

Jo et Averell partagent la même instance de Butin

Partage entre Thread : partage d'un objet Runnable

Cette fois-ci on va partager un objet Runnable entre deux threads qui s'exécutent en parallèle :

```
class Dalton implements Runnable {  
    private Butin leMagot;  
    public void run() {  
        Random rand = new Random();  
        int nbBilletsDansPoche = rand.nextInt(500) + 500;  
        // je vide mes poches et j'ajoute au mago !  
        for (int i = 1; i <= nbBilletsDansPoche ; i++) {  
            leMagot.ajouterAuMago(1);  
            System.out.println("[ " + Thread.currentThread().getName() + " ]:" +  
                               leMagot.getMontant());  
        }  
    }  
}
```

Partage entre Thread : partage d'un objet Runnable

Cette fois-ci on va partager un objet Runnable entre deux threads qui s'exécutent en parallèle :

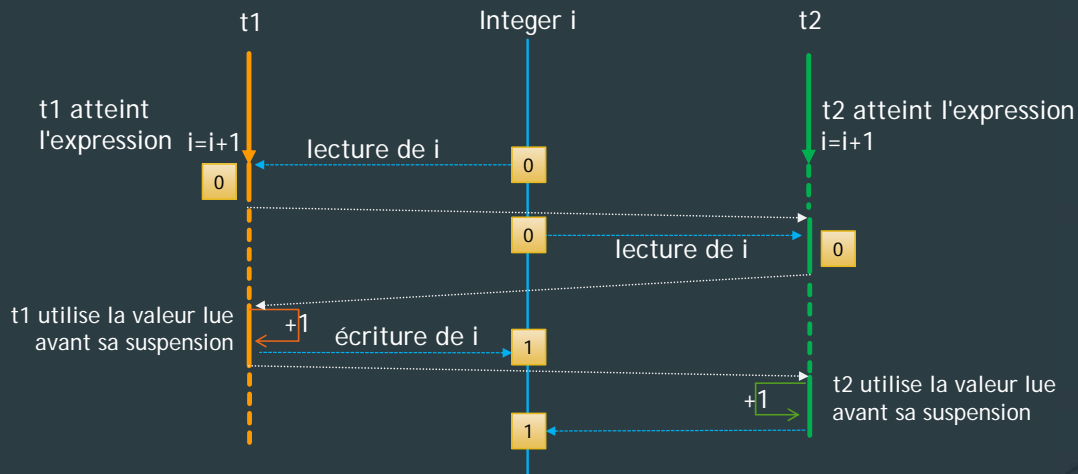
```
public static void main(String[] args) {  
    Dalton daltonType = new Dalton(0); // valeur initiale du butin  
    Thread jo = new Thread(daltonType);  
    jo.setName("Jo");  
    Thread averell = new Thread(daltonType);  
    averell.setName("Averell");  
    jo.start();  
    averell.start();  
}
```

Jo et Averell exécutent le même objet Runnable

Le partage et ses ennuis : "*race condition*"

- ▶ MAIS ... les exemples précédents passent sous silence un risque important du partage : les *race condition*
- ▶ Exemple :
 - deux Thread partagent un entier *i* et arrivent simultanément sur l'expression *i = i+1*
 - *i=i+1* c'est : 1) lire *i* en mémoire, 2) lui ajouter 1, 3) écrire cette nouvelle valeur en mémoire pour *i*
 - Ca fait 3 étapes : pas certain de ne pas être interrompu entre le 1) et le 3) par un thread qui fait le même travail sur le même *i* !

Partage des données : *race condition*



Partage des données : "*race condition*"

- La seule solution c'est de faire en sorte que l'accès à la variable soit exclusif => variable sécurisée vis-à-vis des threads : **Thread Safe**
- Nous verrons cet aspect plus tard car il nécessite un verrou qui interdit "deux accès simultanés" à une même variable (synchronisation qui fait l'objet d'une prochaine étape)



A noter que cela revient à interdire le parallélisme alors que les thread servent à faire du ... parallélisme => à méditer pour le WE

Le partage et ses ennuis : *thread paresseux*

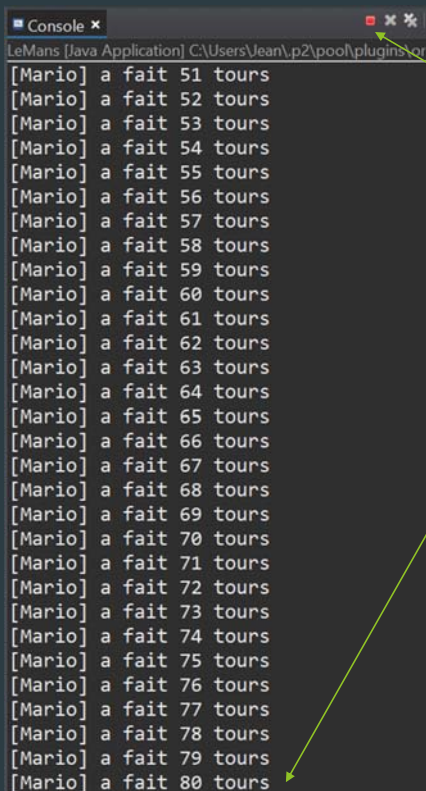
- ▶ un autre problème qui peut survenir est lié à une certaine paresse des thread qui, lorsqu'ils accèdent à une variable partagée ne vont pas toujours récupérer la dernière version en mémoire mais utilise leur copie locale !!
- ▶ *"Oui, je suis en retard car mon agenda ne s'est pas synchronisé donc je n'ai pas la version sur le serveur où le cours a été avancé de 30 minutes"*
- ▶ Ben les thread ... c'est pareil

Exemple : *les 240 tours du Mans*

```
public class Kart extends Thread {  
  
    Kart predecesseur; ← à noter que l'on peut aussi  
    private int nbToursAfaire;      partager des thread  
    private int nbToursParcours;  
  
    public Kart(Kart pred, int todo) {  
        this.nbToursAfaire = todo;  
        this.predecesseur = pred;  
        this.nbToursParcours = 0;  
    }  
  
    public void run() {  
        if (this.predecesseur != null) {  
            while (predecesseur.nbToursParcours < 79) { /*on attend son tour*/ }  
        }  
        for (int i = 1; i <= nbToursAfaire; i++) {  
            nbToursParcours++;  
            System.out.println "[" + getName() + "] a fait " + nbToursParcours + " tours");  
        } }  
}
```

Exemple : *les 240 tours du Mans*

```
public class LeMans {  
  
    public static void main(String[] args) {  
  
        Kart mario = new Kart(null, 80);  
        mario.setName("Mario");  
        Kart luigi = new Kart(mario, 80);  
        luigi.setName("Luigi");  
        Kart fangio = new Kart(luigi, 80);  
        fangio.setName("fangio");  
  
        mario.start();  
        luigi.start();  
        fangio.start();  
    }  
}
```



```
LeMans [Java Application] C:\Users\Jean\p2\pool\plugins\on  
[Mario] a fait 51 tours  
[Mario] a fait 52 tours  
[Mario] a fait 53 tours  
[Mario] a fait 54 tours  
[Mario] a fait 55 tours  
[Mario] a fait 56 tours  
[Mario] a fait 57 tours  
[Mario] a fait 58 tours  
[Mario] a fait 59 tours  
[Mario] a fait 60 tours  
[Mario] a fait 61 tours  
[Mario] a fait 62 tours  
[Mario] a fait 63 tours  
[Mario] a fait 64 tours  
[Mario] a fait 65 tours  
[Mario] a fait 66 tours  
[Mario] a fait 67 tours  
[Mario] a fait 68 tours  
[Mario] a fait 69 tours  
[Mario] a fait 70 tours  
[Mario] a fait 71 tours  
[Mario] a fait 72 tours  
[Mario] a fait 73 tours  
[Mario] a fait 74 tours  
[Mario] a fait 75 tours  
[Mario] a fait 76 tours  
[Mario] a fait 77 tours  
[Mario] a fait 78 tours  
[Mario] a fait 79 tours  
[Mario] a fait 80 tours
```

Dans certaines exécutions :

Mario a terminé et Luigi attend toujours !

Le thread Luigi n'a pas mis à jour "sa" variable
nbToursParcours de Mario



En la déclarant *volatile*, on résout le problème

Le partage et ses ennuis : *thread paresseux*

```
[Mario] a fait 1 tours  
[Mario] a fait 2 tours  
...  
[Mario] a fait 68 tours  
[Mario] a fait 69 tours  
[Mario] a fait 80 tours  
...  
[Luigi] a fait 1 tours  
[Luigi] a fait 2 tours  
...  
[Luigi] a fait 80 tours  
[fangio] a fait 1 tours  
[fangio] a fait 2 tours  
...  
[fangio] a fait 79 tours  
[fangio] a fait 80 tours
```

```
public class Kart extends Thread {  
  
    Kart predecesseur;  
    private int nbToursAfaire;  
    private volatile int nbToursParcoursus;
```



rafraichir la variable avant chaque accès